James Sullivan
45299289

Winter Research Report

Introduction

In this project, I was tasked with exploring ways to generate rely-guarantee conditions for concurrent programs automatically. Recent approaches such as (reference) have achieved this with some success using an iterative strongest-postcondition-based technique. As such, I aimed to develop either a non-iterative, or weakest-precondition-based technique, to deliver greater efficiency or completeness.

In the end, I found this problem to be extremely challenging, yet rewarding to explore. And, though I could not find a fully-developed solution, I did explore many different kinds of approaches, as well as some useful conjectures, which may provide a springboard for future development.

I would like to thank the winter research team, in particular Graeme Smith, Kirstin Winters and Nicholas Coughlin, for their tremendous assistance and guidance throughout my project.

Notation

A thread i is represented with t_i.

A rely condition of t_i is represented as R_i.

A guarantee condition of t_i is represented as G_i.

p represents some predicate, and $P(x)$ represents some predicate of x.

$R : p$ is equivalent to $R = \{p\}$, and similar for G.

$\bigwedge_{i=x}^{n} P(i)$ is equivalent to $P(x) \wedge P(x+1) \wedge \ldots \wedge P(n)$ and similar for $\bigvee_{i=x}^{n}$.

p' is equivalent to $primed(p)$; that is, p where all global variables in p are primed.

A High-Level Explanation of the Problem

The Core Problem

It would be nice to simply find a strongest G_1 and G_2 and a weakest R_1 and R_2, check for compatibility and call it a day. However, the core problem of this project is that the strength of G_i is proportional to the strength required of R_i such that G_i is stable under R_i. Likewise, the weakness of R_i is proportional to the weakness required of G_i such that G_i is

stable under R_i. More on this relationship between a thread's own rely and guarantee can be found under Conjecture 1 & 2.

The SMU Approach

Le et. al from the Singapore Management University (SMU) [1] devised a relatively successful strongest-postcondition-based approach.

First, a strongest guarantee is calculated for both threads (with $R_1 = G_2$ and $R_2 = G_1$ always set as standard) by conducting a strongest postcondition analysis, and then forming the guarantee based on the possible program state changes, where a 'state' is a predicate generated at a particular line of code.

Now the R_1 (generated by calculating G_2 in this way) will have some associated strongest guarantee required for stability under itself, which is unknown at this point. That is, a maximum threshold of strength of G_1 such that the rely can provide stability of the thread. Since G_1 is initially very strong, it is likely to exceed this threshold.

As such, G_1, R_2 is weakened the minimal amount so that G_1 is weaker than or equal to this threshold, by disjoining counterexamples (i.e. states causing instability) to G_1. However, this weakening of R_2 (recall that $G_1 = R_2$ is always maintained) will weaken the strongest guarantee required for stability under R_2, possibly causing G_2 to be stronger than this threshold. We therefore have to weaken G_2, which may in turn require further weakening of G_1, and so forth until either no more weakening is required (in which case the program is verified with the resulting R, Gs) or the rely conditions cannot be weakened any further without permanently destabilising the threads, regardless of guarantee conditions (represented as a failure to find any more counterexamples).

A More Mathematical Description of the Problem

Let $x \ \alpha \ y$ denote "the strength of x is proportional to the strength of y". Let G'_i denote the strongest guarantee required for stability in thread i. We aim to find guarantee and rely conditions such that:

$$\left(G'_1 \implies R_2, G_1\right) \wedge \left(G'_2 \implies R_1, G_2\right)$$

where $R_1 \ \alpha \ G'_1$ and $R_2 \ \alpha \ G'_2$.

One can see that weakening R_2, G_1 to satisfy this predicate may weaken G'_2, potentially necessitating weakening R_1, G_2, which in turn potentially weakens G'_1, potentially invalidating the R_2, G_1 that was initially set, and requiring another iteration of this weakening. Finding out how exactly this rather vague proportionality of $R_1 \ \alpha \ G'_1$ and

$R_2 \; \alpha \; G'_2$ is defined for two given threads is, I suspect, key to finding a non-iterative solution.

Likewise, let R'_i denote the weakest rely required for stability in thread i. Then we can also define the problem as:

$$\left(R_1, G_2 \implies R'_1\right) \wedge \left(R_2, G_1 \implies R'_2\right)$$

where $G_1 \; \alpha \; R'_1$ and $G_2 \; \alpha \; R'_2$.

Conjectures

Conjecture 1

$(R_i \wedge \neg G_i) \in U$

where U represents the set of destabilising states. A destabilising state is one which invalidates the thread when induced by the environment.

For example, for $R : x' = 1 \vee x' = 2$ and $G : x' = 1$, we have $\{x' = 2\} \in U$.

This conjecture is likely not true, for $(R_i \wedge \neg G_i) = false \iff R_i \implies G_i$. Since we also need $(G_1 \implies R_2) \wedge (G_2 \implies R_1)$ for compatibility, this necessitates $R_1 = R_2 = G_1 = G_2$. Thus, $R_i \wedge \neg G_i$ do not necessarily represent unstable states, though it would be interesting to explore counterexamples to more precisely define the relationship between a thread's own rely and guarantee.

Conjecture 2

If $R_i \implies G_i$ then t_i is stable under R_i.

That is, $R_i \implies G_i$ is sufficient, though not necessary for stability.

Conjecture 3

Since $p \wedge R \implies primed(p)$ defines the stability of predicate p under rely R, there are two weakest strengthenings we can induce if stability cannot be proven. They are:

1. $p : p \wedge (R \implies primed(p))$
2. $R : R \wedge (p \implies primed(p))$

Both methods are equivalent to simply conjoining the negation of the counterexample $p \wedge R \wedge \neg primed(p)$. For example, for strengthening p:

$p : p \wedge \neg(p \wedge R \wedge \neg primed(p))$

$p : p \wedge (\neg p \vee \neg R \vee primed(p))$

$p : p \wedge (R \implies primed(p))$

Strengthening p like this may result in primed variables ending up in p, which is possibly not ideal. In addition, a possible optimisation, suggested by Nick (something), which he calls "stabilising p", is:

3. $p : R \implies primed(p)$

This seems to be an often necessary step for proving valid rely conditions via $wpifRG$ proofs.

Conjecture 3

The strongest guarantee for a program can be found via the following procedure:

1. Conduct a strongest postcondition analysis of the program, keeping track of the predicates produced $\{p_1, p_2, \ldots, p_n\}$ at each line.
2. Construct the guarantee condition:
   $$\bigwedge_{i=1}^{n} p_i \implies \left(\bigvee_{j=i}^{n} p_j' \right)$$
   where $p' \equiv primed(p)$.

This guarantee condition is reflexive and transitive, as each state can transition to itself, or any of the following states. Exceptions are for loops.

Conjecture 3-1

A strongest postcondition analysis is required to generate a strongest guarantee for a program.

Conjecture 4

The weakest rely for a program can be found via the following procedure:

1. Conduct a weakest precondition proof of the program, keeping track of the predicates produced $\{p_1, p_2, \ldots, p_n\}$ at each line.
2. Construct the rely condition:
   $$\bigwedge_{i=1}^{n} p_i \implies p_i'$$
   where $p' \equiv primed(p)$.

See appendix 1 for an encouraging example of this method.

This method was derived by the following procedure:

1. Begin a $wpifRG$ proof of the thread, with $G : true$ and $R : true$.
2. When the thread is shown to be unstable at a precondition p (i.e. $stable_R(p) \neq true$), strengthen R by conjoining $p \implies p'$, as per conjecture 3, and continue.

Assuming that each predicate will be unstable, we will derive the specified weakest rely. But there are some potential problems with this approach.

Firstly, we may not require $stable_R(p) = true$ at each precondition to prove stability. We may simply end up with an extra predicate that can be satisfied by an earlier line of code.

Secondly, we are only strengthening R at each unstable predicate, but we can also instead strengthen the predicate itself to stabilise it under R via conjecture 3. Figuring out when to

strengthen R and when to strengthen the predicate is potentially an important and difficult question for weakest-rely-based strategies.

Finally, this weakest rely, as demonstrated in appendix 1, may contain local variables. So far, I have not determined a strategy for getting rid of them, but one could start with abstracting them into $R : \forall v,\ P(v)$ where $P(v)$ represents the old rely condition containing local variable v.

Conjecture 4-1

A weakest precondition analysis is required to generate a weakest rely for a program.

Conjecture 5

All rely and guarantee conditions can be expressed as the conjunction of state changes under conditions for particular global variables.

$$\bigwedge_{i \in Global} \left(i' = i \vee \bigvee_{p \in CSC(i)} p \right)$$

where $CSC(i)$ represents all Conditional State Changes of i. Each conditional state change $p_k \in CSC(i)$ takes the form:

$$\left(\bigvee_{s \in S_k} i' = s \right) \wedge \bigwedge_{c \in C_k} c$$

where S_k represents the set of all states that variable i can change to, under the associated conditions C_k.

This was rather a mouthful so here is an example.

$$R_1, G_2 : \left(x' = x \vee (x' \geq x \wedge z = 1) \vee (x' \leq x \wedge z = 0) \right) \wedge (z' = z \vee true)$$

In this example, x can experience a state change of increasing under the associated condition $z = 1$, and can decrease when $z = 0$. Also, z is unbound.

Notice that $x' \geq x$ is the same as $x' = x \vee x' = x + 1 \vee \ldots$ This represents the $\left(\bigvee_{s \in S_k} i' = s \right)$ part of the equation. I have not determined whether this part needs to be reflexive, as we already have $i' = i$ as a disjunction to all conditional state changes $p \in CSC(i)$.

Disproven Conjectures

Disproven Conjecture 1

The weakest rely for a program can be found via the following procedure:

1. Conduct a strongest postcondition analysis of the program, keeping track of the predicates produced $\{s_1, s_2, \ldots, s_n\}$ at each line.
2. Conduct a weakest precondition proof of the program, keeping track of the predicates produced $\{w_1, w_2, \ldots, w_n\}$ at each line.
3. Construct the rely condition:
   $$\bigwedge_{i=1}^{n} s_i \implies w_i'$$

When trying to prove stability in a $wpifRG$ proof for the predicate w_i, since s_i will be stronger than w_i, we will not be able to show that $w_i \wedge R \implies w_i'$ where $R : s_i \implies w_i'$. We in fact need R to be strengthened to $R : w_i \implies w_i$.

Disproven Conjecture 2

The weakest rely for a program can be found via the following procedure:

1. Conduct a strongest postcondition analysis of the program, keeping track of the predicates produced $\{s_1, s_2, \ldots, s_n\}$ at each line.
2. Construct the rely condition:
   $$\bigwedge_{i=1}^{n} s_i \implies s_i'$$

One might choose this approach by recognising that strengthening p in the equation $p \wedge R \implies p'$ will, as much as possible, reduce the necessity to strengthen R in attempting to generate a weakest R such that the program is stable. However, this thinking is flawed, as a strengthening of p will also result in a strengthening of p'. Thus, the resulting rely condition from this strategy may require other threads to maintain states s_i' that are not strictly necessary to maintain for stability.

Attempted Approaches

Iterating From a Strongest Rely

Start with

$R_1 : \forall x \in Global,\ x' = x$

This immediately gives us reflexivity that we can maintain throughout all iterations by disjoining the extra conditional state changes (see conjecture 5) we need after each iteration. It is also necessarily transitive and stability-providing.

Now, with $G_2 = R_1$, check if G_1 holds for t_2. If not, find a strongest G_2 and once again set $R_1 = G_2$. We can find this strongest G_2 by using the strategy in conjecture 3.

As one can see, this approach quickly turns into a modified version of the SMU approach of starting with a strongest guarantee, described in the introduction of this report. Although we do have a different way of forming transitivity via conjecture 3, we abandoned this approach early in the project due to the enormous size of the predicates generated by the conjecture 3 strategy, as well as the requirement of a strongest-postcondition approach. However, the potential (and I stress *potential*, as I have not explored this) to avoid approximating a transitive closure by using this strategy may be of interest to those continuing the project.

Iterating From a Weakest Rely

This was the most promising strategy we found, as it does not require a strongest postcondition analysis. Instead, it is essentially the SMU approach in reverse: rather than starting with a strongest G_1, we start with a weakest R_1; rather than weakening G_2 to be the strongest guarantee stable under R_2, we strengthen R_2 to be the weakest rely producing stability for G_2; rather than weakening predicates to compute the final postcondition of the programs, we strengthen predicates to compute the final precondition of the program.

By using a weakest precondition approach, we can potentially reduce the size and complexity of our rely and guarantee conditions - saving computational resources - as well as suggest program preconditions to users upon failed verifications. The following is the most detailed description I can provide for the method at this point:

1. Generate a weakest R_1, G_2 and R_2, G_1 using conjecture 4.
2. Check if R_2 provides stability, given G_2. We do this by assessing the $guar$ and $stable_R$ generated at each line using a $wpifRG$ proof on t_2. I am not sure if these need to evaluate to $true$, or if we can instead come up with a way to determine the validity of the predicates they might simplify to. In determining stability, predicates should be

stabilised wherever possible via the method discussed in conjecture 3.
If there is stability, repeat step 2 for R_1 and G_1. If stability of both threads can be found, then the program is successfully verified with the R, Gs derived at this point.

3. If stability in step 2 cannot be determined for a given R_i, then we need to strengthen R_i the minimum amount for it to produce stability, given G_i. Currently I am not clear on how this can be done, though one possible idea is to first find the predicate p in the $wpifRG$ proof that is causing issues (how to do this exactly is likely a difficult and important question), then strengthen R_i to $R_i \wedge (p \implies p')$ as per conjecture 3, and finally transform it into its transitive closure. There are likely better ways to do this.
4. After strengthening R_i, G_j to produce stability, repeat step 2 for R_j. Continue strengthening R_i and R_j in this fashion until stability is proven for both threads, or R_i, G_j cannot be strengthened anymore, lest G_j become too strong for t_j to maintain.

Combining a Strongest Guarantee and Weakest Rely

A non-iterative approach to this solution may seem idealistic - and it is - but given that we can obtain all the information we need to compute one in constant time, it may just be possible. The extent of this information may simply be, for both threads:

1. The strongest guarantee G_i, possibly containing information about all program states and state transitions.
2. The weakest rely R_i.
3. The strongest guarantee G'_i that provides stability under a rely equivalent to the other thread's strongest guarantee G_j.
4. The weakest rely R'_i that provides stability given a guarantee equivalent to the other thread's weakest rely R_j.

Given, $R_i, R_j, G_i, G_j, R'_i, R'_j, G'_i, G'_j$, we may be able to derive a formula that can compute the rely and guarantee conditions in constant time, by using facts of the likes of conjectures 1 and 2. Even if we fail to derive such a formula, more research on such facts may allow us to further optimise other methods.

Appendix

Appendix 1: Weakest Rely of Seqlock

In the Seqlock read program, the weakest precondition following the line `r2 := x` is $z = r1 \implies \Gamma_{r2}$. This will generate this portion of the rely condition:

$$R : \ldots \wedge \Big(\big(z = r1 \implies \Gamma_{r2} \big) \implies \big(z' = r1 \implies \Gamma_{r2} \big) \Big) \wedge \ldots$$

This portion is equivalent to:

$$z \neq r1 \implies z' \neq r1$$

which is indeed the weakest thing the program relies on to be stable. Further research should verify whether the rest of the rely condition generated simplifies down to just this portion.

References

[1] https://ieeexplore.ieee.org/document/9376187